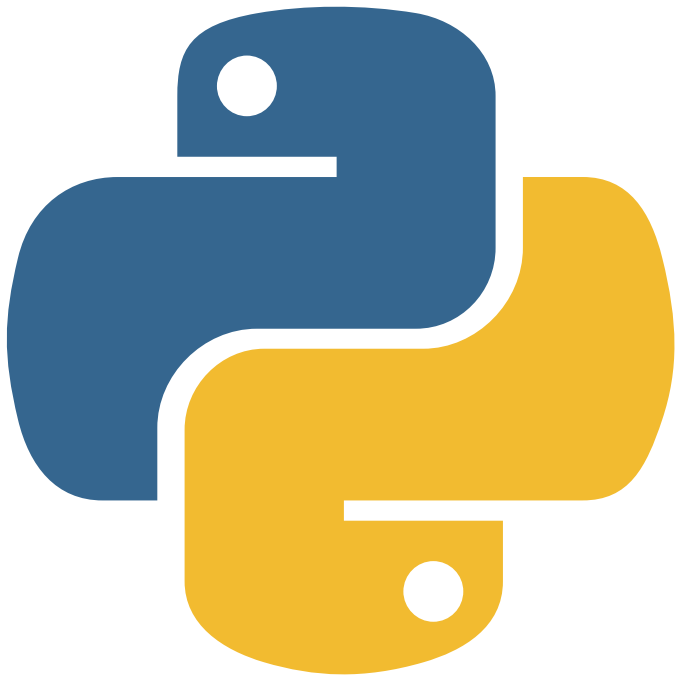
Day 6 Python Basics to Advanced

Day 6 : Lists and Tuples

Today's Topics:

- **list and Methods**
- **Indexing and Slicing**
- **List Comprehension**
- **Tuples**
- **Tuples packing,and unpacking**



By @Curious_.programmer

Lists and Tuples

Lists and tuples in Python both store collections of items. Lists are mutable, allowing changes like adding or removing elements, and use square brackets (`[ ]`).

Tuples are immutable, meaning they can't be modified after creation, and use parentheses (`( )`). Use lists for dynamic data and tuples for fixed collections.

Lists and Methods

Lists in Python are ordered, mutable collections used to store multiple items in a single variable. They are defined using square brackets ([ ]) and can contain elements of different data types, such as integers, strings, or even other lists.
You can add, remove, or modify elements within a list, making them highly versatile for managing dynamic datasets.

Key Features:

- Mutable: Elements can be changed, added, or removed.
- Ordered: Elements have a defined order, and you can access them using indices, starting from 0.

Basic Operations:

- **Create a list:** my_list = [1, 2, 3, 4]
- **Access elements:** my_list[0] (returns 1)
- **Modify elements:** my_list[1] = "new_value"
- **Append elements:** my_list.append(5)
- **Remove elements:** my_list.remove(2) (removes the first occurrence of 2)

1. append()

Adds a single element to the end of the list.
Syntax: list.append(element)

2. extend()

Extends the list by appending elements from an iterable (like another list).
Syntax: list.extend(iterable)

3. insert()

Inserts an element at a specified position in the list.
Syntax: list.insert(index, element)

4. remove()

Removes the first occurrence of a specified element from the list.
Syntax: list.remove(element)

5.pop()

Removes and returns the element at a specified position (index). If no index is specified, it removes and returns the last element.

Syntax: list.pop(index)

Indexing

Indexing in Python refers to accessing individual elements within a sequence, such as a list, tuple, string, or other iterable objects.

Each element in a sequence is assigned a numerical index, starting from 0 for the first element and increasing by 1 for each subsequent element.

Key Points:

1.Positive Indexing:

- The first element has an index of 0.
- The second element has an index of 1, and so on.

2. Negative Indexing:

- Allows you to access elements from the end of the sequence.
- The last element has an index of -1, the second last is -2, and so on.

3. Indexing in Strings:

Indexing works similarly with strings, where each character has an index.

4. Out-of-Range Index:

Accessing an index that is beyond the length of the sequence raises an IndexError.

Slicing

Slicing in Python is a technique used to access a subset of elements from sequences like lists, tuples, or strings.

It allows you to retrieve a portion of the sequence by specifying a start, stop, and optional step index.

- start: The index where the slice begins (inclusive). If omitted, it defaults to the beginning of the sequence (0).
- stop: The index where the slice ends (exclusive). If omitted, it defaults to the end of the sequence.
- step: The step size or interval between elements in the slice. If omitted, it defaults to 1.

List Comprehension

List comprehension is a concise and powerful feature in Python that allows you to create lists in a single line of code.

It combines the process of creating and populating a list with an expression and, optionally, one or more loops and conditions.

Syntax:

```
[expression for item in iterable if condition]
```

- expression: The value or operation applied to each item.
- item: The variable representing the current element in the iteration.
- iterable: The collection (like a list, tuple, or range) that you are iterating over.
- condition: (Optional) A filter that decides whether the expression should be applied to the current item.

Benefits of List Comprehension:

- Concise: It reduces the lines of code needed to create and populate lists.
- Readable: Once you get familiar with the syntax, it can be easier to read and understand.
- Efficient: It often runs faster than traditional for-loop approaches due to its optimized implementation.

Tuples and their Immutabilty

Tuples in Python are ordered collections of items, similar to lists, but with one key difference: tuples are immutable. This means that once a tuple is created, its elements cannot be modified, added, or removed. This immutability makes tuples useful for representing fixed data that should not change throughout the program.

Key Features of Tuples:

Ordered: Like lists, tuples maintain the order of elements, and you can access elements by their index.

Immutable: Once a tuple is created, you cannot change its content. This includes:

Modifying elements: You cannot change the value of any item in the tuple.

Adding elements: You cannot append or insert new items.

Removing elements: You cannot remove items from a tuple.

Defined with Parentheses: Tuples are created by placing a sequence of values separated by commas inside parentheses (()).

Tuples Packing and Unpacking

Tuple packing and unpacking are two related concepts in Python that make working with tuples more convenient and intuitive.

Tuple Packing:

Tuple packing refers to the process of assigning multiple values to a single tuple variable.

When you place multiple values separated by commas into a variable, Python automatically packs them into a tuple.

Tuple Unpacking:

Tuple unpacking is the reverse process, where the values in a tuple are extracted and assigned to individual variables.

The number of variables on the left must match the number of elements in the tuple.

All Coding Handwritten Notes

for Handwritten Notes Search
“CodeWithCurious Store” on Google

Search

CodeWithCurious Store

@curious_.programmer
CodeWithCurious.com

500+ Projects With Source Code

Scan & Download 👇

@curious_.programmer

CodeWithCurious.com